eCube: Hypercube Event for Efficient Filtering in Content-Based Routing

Eiko Yoneki and Jean Bacon

University of Cambridge Computer Laboratory
Cambridge CB3 0FD, United Kingdom
firstname.lastname@cl.cam.ac.uk

Abstract. Future network environments will be pervasive and distributed over a multitude of devices that are dynamically networked. The data collected by pervasive devices (e.g. traffic data, CO_2 values) provide important information for applications that use such contexts actively. Future applications of this type will form a grid over the Internet to offer various services and such a grid requires more selective and precise data dissemination mechanisms based on the content of data. Thus, a smart data/event structure is important. This paper introduces a novel event representation structure, called *eCube*, for efficient indexing, filtering and matching events. We show experimental results that demonstrate the powerful multidimensional structure and applicability of *eCube* over an event broker grid formed in peer-to-peer networks.

1 Introduction

We envision that future network environments will be pervasive, decentralised and distributed over a multitude of devices that are dynamically networked, carried by people and embedded in everyday-life. The stationary and pervasive devices will interact and exchange information in highly dynamic environments in a peer-to-peer (P2P) fashion. Furthermore, the recent emergence of wireless sensor networks (WSNs) has brought a new dimension to data processing, where the sensors are used to gather high volumes of different data (i.e. events from the real world) and to feed them as contexts to a wide range of applications. Such applications are increasingly decentralised and distributed.

In many applications that process data collected from wireless sensor networks (WSNs), the large volume of high-speed data streams makes storage and data processing impossible. This requires a new generation of middleware that can dynamically exchange data in such environments. A service-based approach can provide networked software entities and support them to the users. These include grid services, information services, network services, web services, messaging services and so forth. This is the vision of a service oriented architecture (SOA). Ultimately, the architecture must be an open and component-based structure that is configurable and self-adaptive. A Web service based grid architecture is static and cannot support these diverse subsystems (e.g. ad hoc environments, local clusters, the global Internet) and the bridges that enable them to inter-operate. Service broker grids based on service management are a recent trend in system architecture that supports such platforms. We have reported initial research on SOA-based middleware (see [31] [33] and [34]).

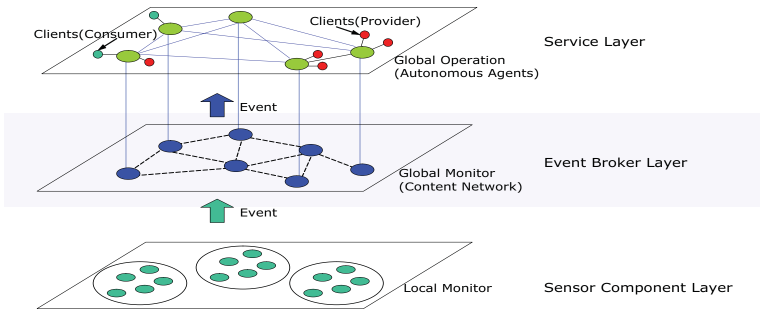


Fig. 1. Service Overlay Architecture

Data management over heterogeneous networks will be crucial. A reactive system incorporating sensing, decision making and acting will be a common application design. Thus, distributed components interact with each other in an event-driven mode. Fig. 1 depicts a scheme of the overlay architecture. The *Sensor Component Layer* performs local and neighbourhood data monitoring, while the *Service Layer* provides services using this information. The *Event Broker Layer* resides between the other two layers to support communication mechanisms. The service overlay architecture must allow information to be integrated at different levels of abstraction, including detailed microscopic examination of specific views of aggregated target behaviour and answers to queries from end users.

The publish/subscribe paradigm fits well with the emerging SOA, in which a distributed application is built using loosely coupled, reusable services. In existing commercial SOA architectures, an *Enterprise Service Bus* (ESB) is provided (e.g. IBM Websphere [18]). The creation of an event broker grid can be easily integrated into SOA. The grid consists of many event brokers, and each broker performs the routing, receiving and sending of events. Brokers can form a group to provide scalability at the cluster level; a group of brokers can then be linked together in a flexible, fault-tolerant and efficient fashion in the publish/subscribe model. Dynamic grid formation is essential, including context-awareness and an infrastructure such as hierarchy and grouping for better performance.

In this paper we focus on data-effective event processing in a publish/subscribe communication paradigm. We identify the necessity of a common event model that can be used for content-based addressing in applications and network components. Application data are influential over data dissemination in pervasive computing. For example, it is important to decide whether to forward data based on spatial information of subscriber nodes when the data is meaningful at a certain location. The state information of the local node may therefore be the event forwarding trigger. Thus, the publish/subscribe model must become more symmetric, so that an event can be disseminated based on the rules and conditions defined by the event itself. The event can then select the destinations instead of relying on the potential receivers' decisions. The symmetric publish/subscribe paradigm brings another level to the data centric paradigm. In the traditional publish/subscribe model, the subscriptions are the complete subset of publications, meaning the subscribers define their subscriptions within the scope of the potential publications. On the other hand, in the symmetric publish/subscribe model, the publications are disseminated based on rules and conditions defined by the publication itself. The publisher rather than the subscriber can choose the destinations. The publishing conditions can be geographical information, physical time, or any local information about potential receivers. For example, epidemic dissemination determines forwarding decisions based on the given parameter of probability and the symmetric dissemination mechanism can define this parameter for each publication individually.

Defining an event without unambiguous semantics requires a fundamental design of event representation. Besides the existing event attributes, event order and continuous context information such as time or geographic location must be incorporated within an event description. We present a multidimensional event representation, the *eCube* structure in RTree (based on [15]) for efficient indexing, filtering, matching, and selective dissemination in publish/subscribe systems. We apply the *eCube* to a content-based publish/subscribe system and experiment with the effect of multidimensional filtering.

This paper's contribution is twofold: First, the *eCube*, a novel event representation structure for efficient indexing, filtering and matching events. Second, we experiment with the *eCube* in a publish/subscribe system in a P2P network. This paper continues as follows: Section 2 and 3 briefly describe the publish/subscribe and event models. Section 4 introduces the *eCube*. Section 5 describes experiments on publish/subscribe systems with the *eCube* in P2P networks. In Section 6, we discuss related works, and Section 7 contains conclusions and future work.

2 Publish/Subscribe Communication

Multi-point asynchronous communication such as publish/subscribe realises the vision of data centric networking that is particularly important for supporting service oriented overlay networks. The data centric approach relies on content addressing instead of host addressing for participating nodes, thus providing network independence for applications. The publish/subscribe paradigm supports decoupling of publishers and subscribers in space and time and integrating scattered WSNs at the edge of wired networks. P2P networks and grids offer promising paradigms for developing efficient distributed systems.

The *Event Broker Layer* depicted in Fig. 1 is important for integrating publish/subscribe systems of various devices under a unified interface. Event brokers can be placed on mobile devices in mobile ad hoc networks to support data sharing among roaming peers and exploit peer resources if possible. Events are at the heart of publish/subscribe systems. Context-awareness allows applications to exploit information on the underlying network context to achieve better performance and group organisation. Information such as availability of resources, battery power, services in reach and relative distances can be used to improve the routing structure of the grid, thus reducing the routing overhead. Use of context-awareness and location awareness are strategies to overcome these limitations.

```
Subscription: (store, (Tesco AND M&S)) resides in Cambridge
Publication:  (store, Tesco), (location, Cambridgeshire)
              where Cambridgeshire > Cambridge
```

Fig. 2. Example Subscription and Publication in Symmetric Publish/Subscribe

2.1 Content-Based Subscription and Routing

Subscription models can be classified into the following three categories: *Topic-based*, *Content-based*, and *Type-based*. In Topic-based publish/subscribe, events are divided into topics, and subscribers subscribe to topics. Common topic-based systems arrange topics in disjoint hierarchies so that a topic cannot have more than one super topic. In Content-based publish/subscribe, a subscription is defined in a constrained manner and evaluated against event content. Type-based publish/subscribe ties events to a programming language type model, database schema, or semi-structured data model (e.g. XML). Content-based routing (CBR) is emerging as a powerful means to provide content-based data dissemination. Applications exploiting CBR can obtain the ability to retain complete control on the filtering patterns. CBR can be at the core of many systems, including publish/subscribe and event notification, distributed databases, and data processing in WSNs.

2.2 Symmetric Publish/Subscribe

In [26], the symmetric nature of publications and subscriptions is discussed. In conventional publish/subscribe systems, if a publication matches a subscription, it is also implied that the subscription matches the publication. A symmetric publish/subscribe system will only send notifications to those subscribers whose subscriptions satisfy the publication. This symmetry allows subscribers to filter out unwanted information and lets publishers target information to a subset of subscribers. As an example, a publisher might want to publish information only to subscribers who are university students. A subscription can contain an active-attribute, which describes the actual information of the subscriber. This is an important concept for publish/subscribe systems to support ubiquitous computing, where subscribers are mobile or the location or distance from a specific object is relevant. In Fig. 2, the subscriber only receives the publication from *Tesco Supermarket in Cambridge*. The event model therefore requires an expression of appropriate attributes for symmetric publish/subscribe.

3 Event Model

In this section, we introduce an event model in an unambiguous way to deal with types of events that require integration of multiple continuous attributes (e.g. time, space, etc.). This attempt is fundamental in establishing a common semantics of events, which will become tokens in a ubiquitous computing scenario. We consider events and event-based services to be of prime importance for ubiquitous computing, and therefore define semantics of events and instances. An event is a message that is generated by an event source and sent to one or more subscribers. Actual event representation may be a structure encoded in binary, a typed object appropriate to a particular object-oriented language, a set of attribute-value pairs, or XML. The basic event definition is described below. Due to space limitations, details of event model is out with the scope of this paper (see [32] for details.).

3.1 Event

The event concept applies to all levels of events from business actions within a workflow to sensing the air temperature. Primitive and composite events are defined as follows:

Definition 1 (Primitive Event). *A primitive event is the occurrence of a state transition at a certain point in time. Each occurrence of an event is called an event instance. The primitive event set contains all primitive events within the system.*

Definition 2 (Composite Event). *A composite event is defined by composing primitive or composite events with a set of operators. The universal event set $\mathbb{E}$ comprises the set of primitive events $\mathbb{E}_p$ and the set of composite events $\mathbb{E}_c$.*

3.2 Typed Event

Definition 3 (Event Type). *The event type describes the structure of an event.*

Event types can be defined by XML with a certain schema; attribute-value pairs with given attributes and value domains; or strongly typed objects. For example, an event notification from a publisher could be associated with a message m containing a list of tuples <type, attribute name(a), value (v)> in XML format, where type refers to a data type (e.g. float, string). Each subscription s is expressed as a selection of predicates in conjunctive form, i.e. $s = \bigwedge_{i=1}^{n} P_i$. Each element P_i of u is expressed as <type; attribute name(a); value range(R) $>$, where $R : (x_i; y_i)$. P_i is evaluated to be true only for a message that contains $< a_i; v_i >$. A message m matches a subscription s if all the predicates are evaluated to be true based on the content of m.

4 *eCube* Hypercube Event

This section presents a multidimensional event representation, the *eCube*, for efficient indexing, filtering, and matching. These operators are fundamental for events and influences a higher-level event dissemination model. There are various data structures and access methods for multidimensional data, and an overview and comparative analysis are presented in [8] [11] [1]. Choosing the indexing structure is complex and has to satisfy the incremental way of maintaining the structure and range query capability. We carefully investigated the UB-tree and RTree structures. The UB-tree is designed to perform multidimensional range queries [3]. It is a dynamic index structure based on a BTree and supports updates with logarithmic performance and space complexity O(n). The RTree is widely used for spatio-temporal data indexing, and it supports dynamic tree splitting and merging operations. Thus, we have chosen RTree to represent multidimensional events and event filtering, where events require dynamic operations.

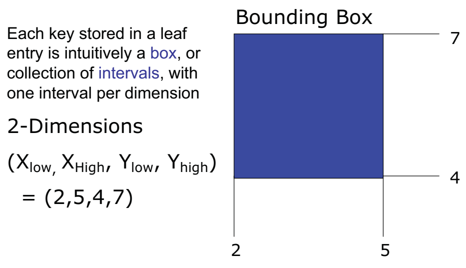

Each key stored in a leaf entry is intuitively a box, or collection of intervals, with one interval per dimension
2-Dimensions
(Xlow, XHigh, Ylow, Yhigh) = (2,5,4,7)
Bounding Box
7
4
2
5

Fig. 3. Minimum Boundary Rectangle

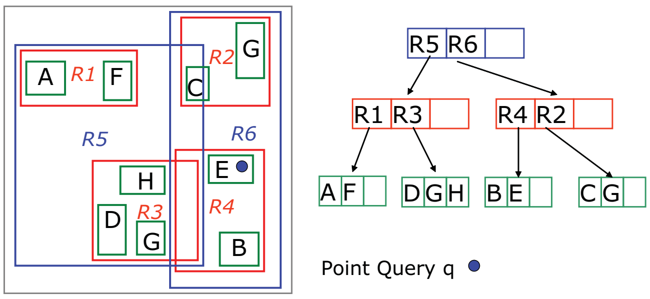


Fig. 4. RTree Structure

4.1 RTree

An RTree [15], extended from a B^+Tree, is a data structure that can index multidimensional information such as spatial data. Fig. 3 shows an example of 2-dimensional data. An RTree is used to store minimum boundary rectangles (MBRs), which represent the spatial index of an n-dimensional object with two n-dimensional points. Similar to BTrees, RTrees are kept balanced on insert and delete, and they ensure efficient storage utilisation.

Structure. An RTree builds a MBR approximation of every object in the data set and inserts each MBR in the leaf level nodes. Fig. 4 illustrates a 3-dimensional RTree; rectangles A-F represent the MBRs of the 3-dimensional objects. The parent nodes, R5 and R6, represent the group of object MBRs. When a new object is inserted, a cost-based algorithm is performed to decide in which node a new object has to be inserted. The goals

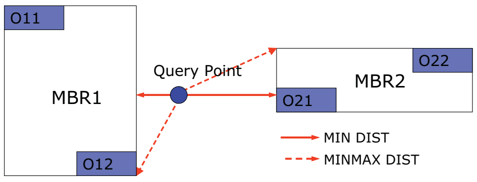


Fig. 5. Nearest Neighbour Search

of the algorithm are to limit the overlap between nodes and to reduce the dead-space in the tree. For example, grouping objects A, C, and F into R5 requires a smaller MBR than if A, E, and F were grouped together instead. Enforcing a minimum/maximum number of object entries per node ensures balanced tree formation. When a query object searches the tree for the intersection operation, the tree is traversed, starting at the root, by passing each node where the query window intersects a MBR. Only object MBRs that intersect the query MBR at the leaf-level have to be retrieved from disk. A BTree may require a single path through the tree to be traversed, while an RTree may need to follow several paths, since the query window may intersect more than one MBR in each node. MBRs are hierarchically nested and can overlap. The tree is height-balanced; every leaf node has the same distance from the root. Let M be the number of entries that can fit in a node and m the minimum number of entries per node. Leaf and internal nodes contain between m and M entries. As items are added and removed, a node might overflow or underflow and require splitting or merging of the tree. If the number of entries in a node falls under the m bound after a deletion, the node is deleted, and the rest of its entries are distributed among the sibling nodes.

Each RTree node corresponds to a disk page and an n-dimensional rectangle. Each non-leaf node contains entries of the form $(ref, rect)$, where ref is the address of a child node and $rect$ is the MBR of all entries in that child node. Leaves contain entries of the same format, where ref points to an object, and $rect$ is the MBR of that object.

Search. Search in an RTree is performed in a similar way to that in a BTree. Search algorithms (e.g. intersection, containment, nearest) use MBRs for the decision to search inside a child node. This implies that most of the nodes in the tree are never *touched* during a search. The average cost of search is $O(\log n)$ and the worst case is $O(n)$. Different algorithms can be used to *split* nodes when they become full. In Fig. 4, a point query *q* requires traversing R5, R6 and child nodes of R6 (e.g. R2 and R4) before reaching the target MBR E. When the coverage or overlap of MBRs is minimised, RTree gives maximum search efficiency.

For *nearest neighbour* (NN), the search for point data is based on the distance calculation shown in Fig. 5. Let $MINDIST(P, M)$ be the minimum distance between a query point and a boundary rectangle, and let $MINMAXDIST(P, M)$ be the upper bound of minimum distance to data in the boundary rectangle (i.e. among the points belonging to the lines consisting of MBR, select the one closest to the query point). However, there is no guarantee that the MBR contains the nearest object even if $MINDIST$ is small. In Fig. 5, the smaller $MINDIST$ from the query point is MBR1, while the nearest object of O21 is in MBR2. The search algorithm for *nearest neighbour* is:

1. If the node is a $leaf$, then find NN. If non $leaf$, sort entries by MINDIST to create *Active Branch List* (ABL).
2. if $MINDIST(P, M) > MINMAXDIST(P, M)$ then remove MBR. If the distance from the query point to the object is larger than $MINMAXDIST(P, M)$ then the object is removed (i.e. M contains an object that is closer to P than the object). If the distance from the query point to the object is larger than $MINDIST(P, M)$, then M is removed (i.e. M does not contain objects that are closer to P than the object).
3. Repeat 1 and 2 until ABL is empty.

4.2 Adaptation to Publish/Subscribe

Event filtering in a content-based publish/subscribe system can be considered as querying in a high dimensional space, but applying multidimensional index structures to publish/subscribe systems is still unexplored. Thus, we have both publication and subscription are modelled as *eCubes* in our implementation, where matching is regarded as an intersection query on *eCubes* in an n-dimensional space. Point queries on the *eCube* are transformed into range queries to make use of efficient point access methods for event matching. This corresponds to the realisation of symmetric publish/subscribe, and it automatically provides effective range queries, nearby queries, and point queries.

Traditional databases support multidimensional data indexing and query, when using a query language as an extension of SQL. For example, a moving object database can index and query position/time of tracking objects. Applications in ubiquitous computing require such functions over distributed network environments, where data are produced by publishers via event brokers, and the network itself can be considered as a database. The query is usually persistent (i.e. continuous queries). Stream data processing and publish/subscribe systems address similar problems. Nevertheless, supporting spatial, temporal, and other event attributes with a multidimensional index structure can dramatically enhance filtering and matching performance in publish/subscribe systems. For example, the event of tracking a car, which is associated with changes of position through time, needs spatio-temporal indexing support. GPS, wireless computing and mobile phones are able to detect positions of data, and ubiquitous applications desperately need this data type for tracking, rerouting traffic, and location aware-services.

Both point and range queries can be performed over the *eCube* in a symmetric manner between publishers and subscribers. The majority of publish/subscribe systems consider that subscriptions cover event notifications. We focus on symmetric publish/subscribe, and the case of when event notifications cover subscriptions is therefore also part of the event filtering operation. Thus, typical operations with the *eCube* can be classified into the following two categories:

– **Event Notifications $\subseteq$ Subscriptions:** events are point queries and subscriptions are aggregated in the *eCube*. For example, subscribers are interested in the stock price of various companies, when the price dramatically goes up. All subscribers have interests in different companies, and an event of a specific company's price change will be notified only to the subscribers with the matching subscriptions.
– **Event Notifications $\supseteq$ Subscriptions:** events are range queries and subscriptions are point data. For example, a series of news related to *Bill Gates* is published to the subscribers who are located in New York and Boston. Thus, an attribute indicates the location in the event notification to *New York and Boston*. Subscribers with the attribute *London* will not receive the event.

4.3 Cube Subscription

Events and subscriptions can essentially be described in a symmetric manner with the *eCube*. Consider an online market of music, where old collections may be on sale. Events represent a cube containing 3 dimensions (i.e. Media, Category, and Year). Subscriptions can be:

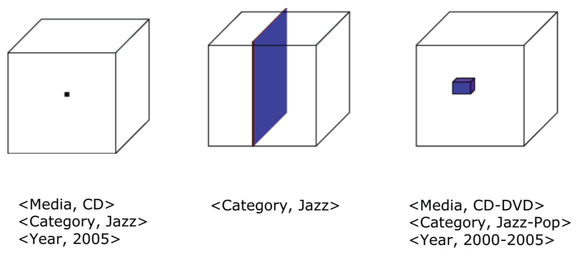


Fig. 6. 3-Dimensional Subscription

Point Query: CDs of Jazz released in 2005
Partial Match Query: Any media of Jazz
Range Query: CDs and DVDs of Jazz and Popular music released between 2000 and 2005

Fig. 6 depicts the 3-dimensional *eCube* and the above subscriptions are shown.

4.4 Expressiveness

We consider event filtering as search in high dimensional data space and introduce a hypercube based filtering model. It is popular to index spatio-temporal objects by considering time as another dimension on top of a spatial index so that a 3-dimensional spatial access method is used. We consider extension to n dimensions, which allows to include any information such as weather, temperature, or interests of the subscribers. Thus, this approach takes advantage of the range query efficiency by using multidimensional indexing. The indexing mechanism with the *eCube* can be used for filtering expression for content-based filtering, aggregation of subscription, and part of the event correlation mechanism. Ultimately, the event itself can be represented as a *eCube* for symmetric publish/subscribe.

Thus, the *eCube* filter uses the geometrical intersection of publications/subscriptions represented in hypercubes in a multidimensional event space. This will provide selective data dissemination in an efficient manner including *symmetric publish/subscribe*. Data from WSNs can be multidimensional and searching for these complex data may require more advanced queries and indexing mechanisms than simply hashing values to construct a DHT so that multiple pattern recognitions and similarities can be applied. Subscribing to unstructured documents that do not have a precise description may need some way to describe the semantics of the documents. Another aspect is that searching a DHT requires the exact key for hashing, while users may not require exact results. This section discusses the expressiveness of query and subscription.

The *eCube* can express these subscriptions and filtering by use of another dimension with time values. A simple real world example for use of the *eCube* can be with geographical data coordinates in 2-dimensional values. A query such as *Find all book stores within 2 miles of my current location* can be expressed in an RTree with the data splitting space of hierarchically nested, and possibly overlapping, rectangles.

4.5 Experimental Prototype

The prototype implementation of RTree is an extension of the Java implementation [16] based on the paper by Guttman [15]. We extended it to become more compact. It currently supports range, point, and nearest neighbour queries. The prototype is a 100KB class library in Java with JDK 1.5 SE. The experiments aim to demonstrate the applicability of an RTree for event and subscription representation.

4.6 Evaluation of *eCube* with Sensor Data

In this section, we show the brief evaluation of the *eCube* addressing the filtering capability. We experiment the *eCube* with live traffic data from the city of Cambridge. Data is gathered from sensors of inductive loops installed at various key junctions across Cambridge and collected every five minutes from raw signal information. Different sizes of data sets are used for the experiments, ranging from 100 to 40,000. The motor-way data from April 3rd 2006 is used, which is transformed into 1-, 3-, and 6-dimensional data with attributes *Date, Day, Time, Location, Flow* and *Occupancy*. The raw data are point data, which are converted to zero size range data so that range queries can be issued against them by the intersection operation. This experiment demonstrates the functionality of RTree and compares the operation with a simple *brute force* operation, where the set of predicates are used for query matching.

Complex range queries directly mapping to real world incidents can be processed such as *speed of average car passing at junction A is slower than at junction B at 1:00 pm on Wednesdays*. It is not easy to show the capability of the *eCube* filtering for expressive and complex queries in a quantitative manner. Thus, experiments focus on the performance of a high-volume range filtering processes.

Dimension Size. Fig. 7 and Fig. 8 show the processing speed of a range query. The X axis indicates the data size with *bytes*; it is not linearly scaled over the entire range. This X axis coordination is same in Fig. 7-11. The sizes of data sets are selected between 100 and 40,000 as seen on the X axis. Two partitions (between 1000 and 5000, and between 10,000 and 40,000) are scaled linearly. This applies to all the experiments, where different data sets are used. An RTree has been created for data of 1, 3, and 6 dimensions.

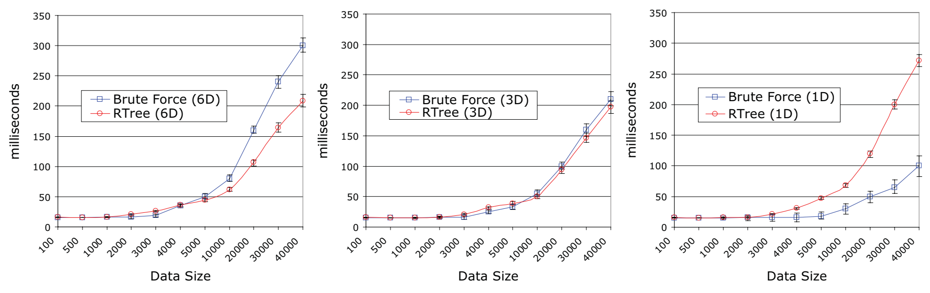


Fig. 7. Single Range Query Operation: RTree vs. Brute Force

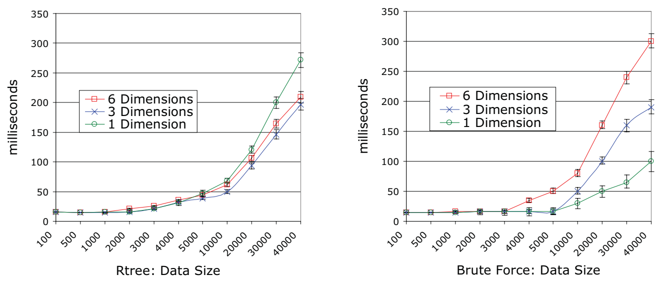


Fig. 8. Single Range Query: Dimensions

The operation using the *brute force* method is also shown, where each predicate is compared with the query. For 1-dimensional data, the use of RTree incurs too much overhead, but the RTree outperforms at increasing numbers of dimensions. The difference in the number of dimensions has little influence over the RTree performance. Thus, once the structure is set, it guarantees an upper bound on the search time.

Matching Time. Fig. 9 and Fig. 10 show average matching operation times for a data entry against a single query. The Y axis indicates *total matching time / number of data items*. Fig. 9 is depicted the comparison between RTree and *Brute Force* within the same dimensional data, while Fig. 10 shows the same experiment results for comparison of different dimensional data. For 1-dimensional data, the use of RTree incurs too much overhead, but increasing dimensions does not affect operation time. In these figures, the X axes are in non-linear scales. The cost of the brute force method increases with increasing dimension of data, which is shown in Fig. 10.

RTree Storage Size. Fig. 11 shows the storage requirement for RTree. The left figure shows storage usage, while the right one shows construction time. The current configuration uses 4096B per block. Since the index may also contain user defined data, there is no way to know how big a single node may become. The same data set is used for the repeating experiments and the standard deviation is therefore 0. The storage manager will use multiple blocks per node if needed, which will slow down performance. There are only few differences with changing dimension size, because the data size in each element is about the same in this experiment. The standard deviation value is $\cong 0$, because the input data for each experiment is identical.

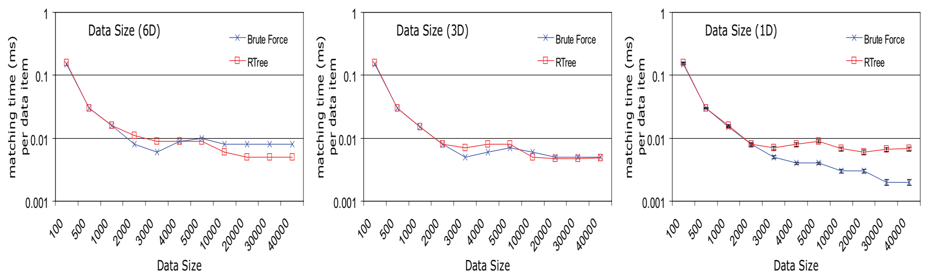


Fig. 9. Single Range Query Matching Time

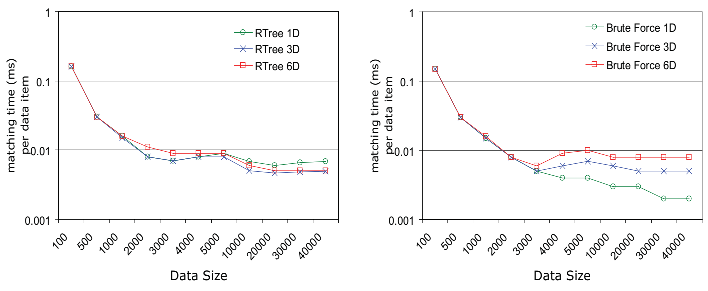


Fig. 10. Matching Time

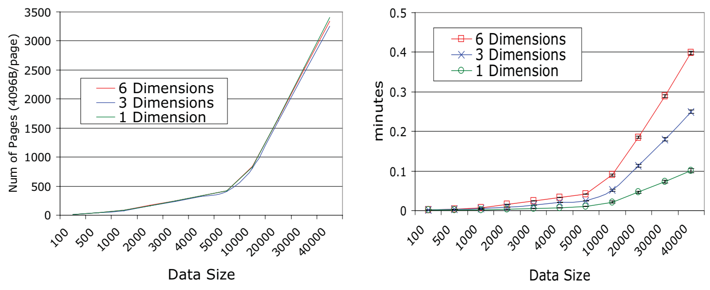


Fig. 11. Construction of RTree

The experiments highlight that RTree based indexing is effective for providing data selectivity among high volumes of data. It gives an advantage for incremental operation without the need for complete reconstruction. These experiments are not exhaustive and different trends of data may produce different results. Thus, it will be necessary to conduct further experiments with various real world data as future work.

RTree indexing enables neighbourhood search, which allows similarity searches. This will be an advantage for supporting subscriptions that do not pose an exact question or only need approximate results. Approximation or summarisation of sensor data can be modelled using this function.

5 Event Broker Grid with *eCube* Filter

We present an extension to a typed content-based publish/subscribe system (i.e. Hermes) with the *eCube* filtering. In content-based publish/subscribe, the *eCube* filter can be placed in the publisher and subscriber edge brokers, or distributed over the networks based on the coverage relationship of filters. If the publish/subscribe system takes rendezvous routing, a rendezvous node needs to keep all the subscriptions for the matching. Multidimensional range queries support selective data to subscribers who are interested in specific data.

Hermes [23] is a typed content-based publish/subscribe system built over Pastry. The basic mechanism is based on the rendezvous mechanism that Scribe uses [7]. Addition-

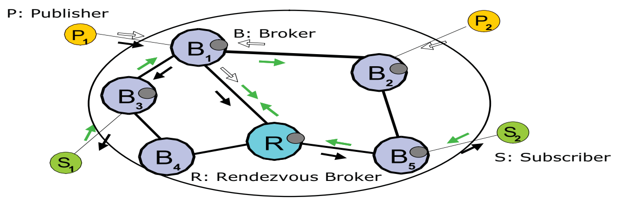


Fig. 12. Content-Based Routing for Publish/Subscribe in Hermes

ally, Hermes enforces a typed event schema providing type safety by type checking on notifications and subscriptions at runtime. The rendezvous nodes are chosen by hashing the event type name. Thus, it extends the expressiveness of subscriptions and aims to allow multiple inheritance of event types. In Hermes, the content-based publish/subscribe routing algorithm is an adaptation of SIENA [6] and Scribe using rendezvous nodes. Both advertisements and subscriptions are sent towards the rendezvous node, and each node en route keeps track. Routing between the publisher, where the advertisement comes from, and the subscriber is created through this process. An advertisement and subscriptions meet in the worst case at the rendezvous node. The event notification follows this routing, and the event dissemination tree is therefore rooted from the publisher node. This will save some workload from the rendezvous nodes.

Fig. 12 shows routing mechanisms for content-based publish/subscribe. Arrows are white for advertisements, light grey for subscriptions, and black for publications. The black arrow from broker 1 to broker 3 shows a shortcut to subscriber 1 that is different from the routing mechanism of Scribe. Subscription 2 in content-based routing travels up to the broker hosting the publisher Fig. 12. Grey circles indicate where filtering states are kept.

5.1 *eCube* Event Filter

In content-based networks such as SIENA [6], the intermediate server node creates a forwarding table based on subscriptions and operates event filtering. Under high event publishing environments, the speed of filtering based on matching the subscription predicates at each server is crucial for obtaining the required performance.

In [25] and [4], subscriptions are clustered to multicast trees. Thus, filtering is performed at both the source and receiver nodes. In contrast, the intermediate nodes perform filtering for selective event dissemination in [21]. In Hermes, a route for event dissemination for a specific event type is rooted at the publisher node through a rendezvous node to all subscribers by constructing a diffusion tree. The intermediate broker nodes operate filtering for content-based publish/subscribe. The filtering mechanism is primitive, with each predicate of the subscription filter being kept independently without any aggregation within the subscriber edge broker. The coverage operation requires a comparison of each predicate against an event notification.

The *eCube* is integrated to subscription filters to provide efficient matching and coverage operations. In the experiments, the effectiveness and expressiveness of typed channels and filtering attributes are compared. The advantages of this approach include efficient range query and filter performance (resource and time).

The balance between typed-channel and content-based filtering is a complex issue. In existing distributed systems, each broker has a multi-attribute data structure to match the complex predicate for each subscription. The notion of weak filtering for hierarchical filtering can be used as summary-based routing (see [30] and [9]), so that the balance between the latency of the matching process and event traffic can be controlled. When highly complex event matching is operated on an event notification for all subscriptions, it may result in too high message processing latency. This prevents reasonable performance of publishing rates to all subscribers. The subscription indexing data structure and filter matching algorithm are two important factors to impact the performance in such environments including filter coverage over the network.

Event filtering in content-based publish/subscribe can provide better performance if similar subscriptions are in a single broker or neighbour brokers. Physical proximity provides low hop counts per event diffusion in the network with a content-based routing algorithm [20]. If physical proximity is low, on the other hand, routing becomes similar to simple flooding or unicasting.

5.2 Range Query

A DHT is not suited for range queries, which makes it hard to build a content-based publish/subscribe system over structured overlay networks. When the subscription contains attributes with continuous values, it becomes inefficient to walk through the entire DHT entries for matching. Range queries are common with spatial data and desirable in geographic-based applications of pervasive computing, such as queries relating to intersections, containment, and nearest neighbours. Thus, *eCube* provides critical functions. However, DHT mechanisms in most of the current structured overlay distribute data uniformly without an exhaustive search. Range queries introduce new requirements such as data placement and query routing in distributed publish/subscribe systems.

5.3 Experiments

The experiment in this section demonstrates a selective and expressive event filter that can be used to provide flexibility to explore the subscriptions. The performance of scalability issues in Hermes is reported in [23] and general control traffic (e.g. advertisement, subscription propagation) are also reported in [27].

Thus, to keep the results independent of secondary variables, only the message traffic for the dissemination of subscriptions is therefore measured. The metrics used for the experiments are the number of publications disseminated in the publish/subscribe system. The number of hops in the event dissemination structure varies depending on the size of the network and the relative locations between publishers and subscribers.

Experimental Setup. The experiments are run on FreePastry [27], a Pastry simulator. Publishers, subscribers and rendezvous nodes are configured with deterministic $node\ ids$, and all the other brokers get $node\ ids$ from Pastry simulations.

One thousand Pastry nodes are deployed. All pastry nodes are considered as brokers, where the Hermes event broker function resides, and the total number of nodes (N=1000) gives average hop counts from the source to the routing destination as

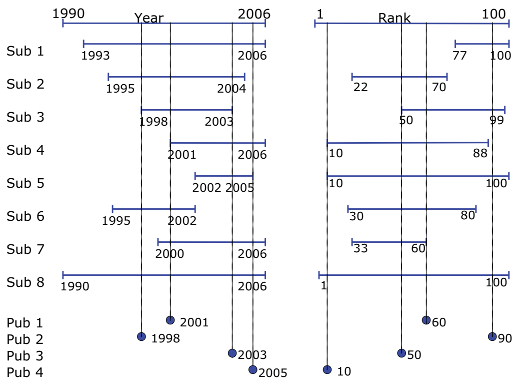


Fig. 13. Subscriptions and Publications

$\log_{24}(1000) \approx 2.5$, where 4 is given as a configuration value. Eight subscribers connect to the subscriber edge brokers individually. The subscriptions are listed in Fig. 13. 1000 publications are randomly created for each event type by a single publisher. This is a relatively small scale experiment, but considering the characteristics of Hermes, where each publisher creates an individual tree combining the rendezvous node, the experiment is sufficient for evaluation.

Subscriptions and publications. A single type CD with two attributes (i.e. *released year* and *ranking*) are used for the content-based subscription filter. In Fig. 13, eight subscriptions are defined with different ranges on two attributes. The publications take the form of a point for the *eCube* RTree. Four different publications are defined and 250 instances of each publication are published: 1000 event notifications are processed

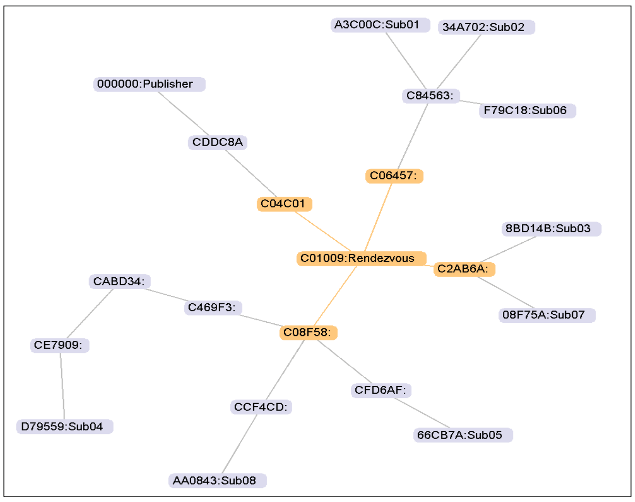


Fig. 14. Pub/Sub System over Pastry

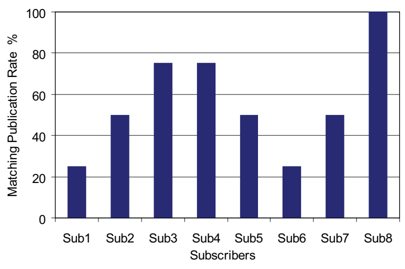


Fig. 15. Matching Rate in Scribe (No Filtering)

in total. Same sets of publications and subscriptions are used in all experiments unless stated otherwise.

Base Case with *eCube* Event Filter. This experiment demonstrates the basic operation of *eCube* event filters. The experiment is operated on Hermes with *eCube* filtering and Scribe, where no filtering is equipped. Fig. 14 shows the logical topology consisting of 8 subscribers (i.e. *Sub01-Sub08*), a publisher, and a rendezvous node along router nodes. Identifiers indicate the addresses assigned by the Pastry simulation.

Fig. 15 depicts the matching publication rate for each subscriber node in the Scribe experiment. With *eCube*s, there are no false positives and subscribers receive only matching publications. It is obvious that filters significantly help to control the traffic of event dissemination.

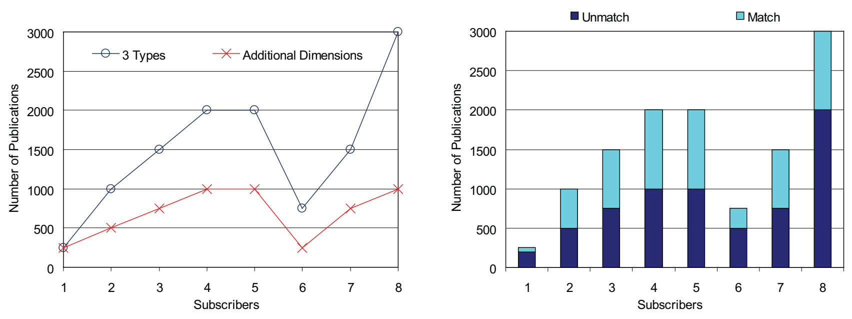
Multiple Types vs. Additional Dimension as Type. When multiple types share the same attributes, there will be two ways: first, defining three predefined types for separated channels and second, defining a single channel with an additional attribute, which distinguish different types.

This experiment operates two settings and compares the publication traffic. In the first scenario, three types are used: $Classic$, $Jazz$, and Pop. Thus, 3 rendezvous nodes are created. All three types share the same attributes. Table 1 shows the defined types along the subscriptions. The publisher publishes 1000 events for each type, 3000 publications in total. Unless there is a super type defined for three types, each type creates an independent dissemination tree and causes multiple traffic.

In Fig. 17, three rendezvous nodes appear for each type. For the second setting, instead of using multiple types, an additional dimension is added to the *eCube*. Fig. 16(a) depicts the total event traffic between two settings. The apparent result shows

Table 1. 3 Types and Matching Subscriptions

Subscriber	Classic	Jazz	Pop
Sub 1	✓	✓	
Sub 2		✓	✓
Sub 3	✓		✓
Sub 4	✓	✓	
Sub 5	✓	✓	✓
Sub 6	✓	✓	✓
Sub 7	✓		✓
Sub 8	✓	✓	✓



(a) Traffic Comparaison (b) Matching Rate (3 types)

Fig. 16. Comparison between Channels on Types vs. Additional Dimensions

significant improvement of the traffic with the additional dimensional approach. Fig. 16(b) shows the matching ratio on received events in the experiment with 3 types.

When different event types are used, which are not hierarchical, separated route construction for each event type is performed for event dissemination. Different types, which may contain the same attributes, may not have a super type. Also super types may contain many other subtypes, of which the client may not want to receive notifications. Thus, additional dimensions on the filtering attributes may be a better approach for flexible indexing. Transforming the type name to the dimension can preserve locality, similarity or even hierarchy. This will provide an advantage for neighbour matching.

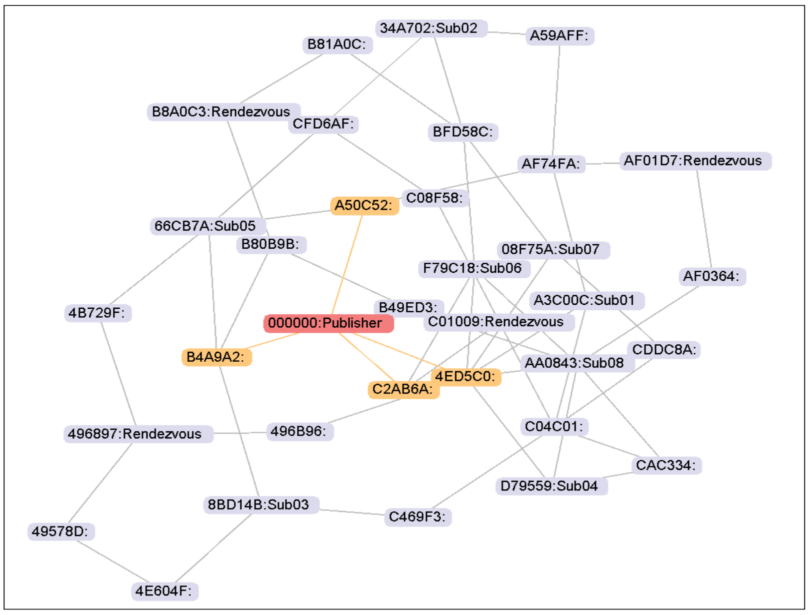


Fig. 17. Publish/Subscribe System with 3 Event Types

The *eCube* filter introduces flexibility between the topic and content-based subscription models. The experiments show that adding additional dimensions in the hypercube filter transforming from $type$ outperforms constructing on individual channel for $type$. Transforming the type name to a dimension can preserve similarity and hierarchy, that automatically provides neighbour matching capability. Further experiments for flexible indexing will be useful future work.

DHT mechanisms contain two contradictory sides: the hash function distributes the data object evenly within the space to achieve a balanced load, whereas the locality information among similar subscriptions may be completely destroyed by applying a hash function. For example the current Pastry intends to construct DHT with random elements to accomplish load balance. Nevertheless similarity information among subscriptions is important in publish/subscribe systems.

6 Related Work

In database systems, multidimensional range query is solved using indexing techniques, and indices are centralised. Recently distributed indexing is becoming popular, especially in the context of P2P and sensor networks. Indexing techniques tradeoff data insertion cost against efficient querying (see [32] for further details).

A similar idea to the *eCube* is CAN-based multicast [24]. In [19], Z-ordering [22] is used for the implementation of CAN multicast. Z-ordering interleaves the bits of each value for each dimension to create a one-dimensional bit string. For matching algorithms, fast and efficient matching algorithms are investigated for publish/subscribe system in [10]. Topic-based publish/subscribe is realised by a basic DHT-based multicast mechanism in [35], [36], [24]. More recently, some attempts on distributed content-based publish/subscribe systems based on multicast have become popular [2], [5], [29]. An approach combining topic-based and content-based systems using Scribe is described in [28]. In these approaches, the publications and the subscriptions are classified in topics using an appropriate application-specific schema. The design of the domain schema is a key element for system performance, and managing false positives is critical for such approach.

Recently, several proposals have been made to extend P2P functionality to more complex queries (e.g. range queries [14], joins [17], XML [12]). [13] describes the Range Search Tree (RST), which maps data partitions into nodes. Range queries are broken to sub-queries corresponding to nodes in the RST. Data locality is obtained by the RST structure, which allows fast local matching. However, sub-queries make the matching process complex.

Our *eCube* demonstrates a unique approach for representing events and can be used in different systems.

7 Conclusions

In this paper, we have introduced *eCube*, a novel event representation structure for efficient indexing, filtering and matching events and have applied it with a typed content-based publish/subscribe system for improvement of event filtering processes.

The experiments show various advantages including efficiency of range queries and additional dimensions in the hypercube filter transforming from $type$. Transforming the type name to a dimension can preserve similarity and hierarchy that automatically provides neighbour matching capability.

We continue to work on a regular expression version of RTree for a better indexing structure. Transformation mechanisms such as a feature extraction process to reduce the number of dimensions may be useful. A series of future work include lightweight versions of indexing structure for supporting resource-constrained devices and fuzzy semantic queries for the matching mechanism. An important aspect is that the values used to index *eCube* will have a huge impact. For example, the use of a locality sensitive hashing value from string data and the current form of the *eCube* filter can both be exploited with the locality property. This will be worthwhile future work.

Acknowledgment. I would like to thank to Derek Murray for valuable comments.

References

1. Ahn, H.K., Mamoulis, N., Wong, H.M.: A survey on multidimensional access methods. Technical report, Utrecht University (2001)
2. Banavar, G., et al.: An efficient multicast protocol for content-based publish-subscribe systems. In: Proc. ICDCS, pp. 262–272 (1999)
3. Bayer, R.: The universal B-tree for multidimensional indexing. Technical Report TUM-I9637, Technische Universitat Munchen (1996)
4. Cao, F., Singh, J.: Efficient event routing in content-based publish-subscribe service networks. In: Proc. IEEE INFOCOM (2004)
5. Carzaniga, A., Rosenblum, D., Wolf, L.: Design and evaluation of a wise-area event notification service. ACM Trans. on Computer Systems 19(3) (2001)
6. Carzaniga, A., Rutherford, M., Wolf, A.: A routing scheme for content-based networking. In: Proc. IEEE INFOCOM (2004)
7. Castro, M., et al.: Scribe: A large-scale and decentralized application-level multicast infrastructure. Journal on Selected Areas in Communication 20 (2002)
8. de Berg, M., et al.: Computational Geometry-Algorithms and Applications. Springer, Heidelberg (1998)
9. Eugster, P., Felber, P., et al.: Event systems: How to have your cake and eat it too. In: Proc. Workshop on DEBS (2002)
10. Fabret, F., Jacobsen, H.A., et al.: Filtering algorithms and implementation for very fast publish/subscribe systems. In: Proc. SIGMOD, pp. 115–126 (2001)
11. Gaede, V., et al.: Multidimensional access methods. ACM Computing Surverys 30(2) (1998)
12. Galanis, L., Wang, Y., et al.: Locating data sources in large distributed systems. In: Proc. VLDB, pp. 874–885 (2003)
13. Gao, J., Steenkiste, P.: An adaptive protocol for efficient support of range queries in DHT-based systems. In: Proc. IEEE International Conference on Network Protocols (2004)
14. Gupta, A., et al.: Approximate range selection queries in peer-to-peer systems. In: Proc. CIDR, pp. 141–151 (2003)
15. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proc. ACM SIGMOD (1984)
16. Hadjueleftheriou, M.: Spatial index, `http://research.att.com/~marioh/spatialindex/index.html.`

17. Harren, M., et al.: Complex queries in DHT-based peer-to-peer networks. In: Proc. Workshop on P2P Systems, pp. 242–250 (2002)
18. IBM. IBM MQ Series (2000), `http://www.ibm.com/software/ts/mqseries/`
19. jxta.org. `http://www.jxta.org/`
20. Meuhl, G., Fiege, L., Buchmann, A.: Filter similarities in content-based publish/subscribe systems. In: Proc. ARCS (2002)
21. Oliveira, M., et al.: Router level filtering on receiver interest delivery. In: Proc. NGC (2000)
22. Orenstein, J., Merrett, T.: A class of data structures for associative searching. In: Proc. Principles of Database Systems (1984)
23. Pietzuch, P., Bacon, J.: Hermes: A distributed event-based middleware architecture. In: Proc. Workshop on DEBS (2002)
24. Ratnasamy, S., et al.: Application-level multicast using content-addressable networks. In: Crowcroft, J., Hofmann, M. (eds.) NGC 2001. LNCS, vol. 2233, Springer, Heidelberg (2001)
25. Riabov, A., Liu, Z., Wolf, J., Yu, P., Zhang, L.: Clustering algorithms for content-based publication-subscription systems. In: Proc. ICDCS (2002)
26. Rjaibi, W., Dittrich, K.R., Jaepel, D.: Event matching in symmetric subscription systems. In: Proc. CASCON (2002)
27. Rowstron, A., Druschel, P.: Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In: Proc. ACM.IFIP/USENIX Middleware, pp. 329–350 (2001)
28. Tam, D., Azimi, R., Jacobsen, H.-A.: Building content- based publish/subscribe systems with distributed hash tables. In: DBISP2P 2004 (2003)
29. Terpstra, W.W., et al.: A peer-to-peer approach to content-based publish/subscribe. In: Proc. Workshop on DEBS (2003)
30. Wang, Y., et al.: Summary-based routing for content-based event distribution networks. ACM Computer Communication Review (2004)
31. Yoneki, E.: Event broker grids with filtering, aggregation, and correlation for wireless sensor data. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2005. LNCS, vol. 3762, pp. 304–313. Springer, Heidelberg (2005)
32. Yoneki, E.: ECCO: Data Centric Asynchronous Communitcation. PhD thesis, University of Cambridge, Technical Report UCAM-CL-TR677 (2006)
33. Yoneki, E., Bacon, J.: Object tracking using durative events. In: Enokido, T., Yan, L., Xiao, B., Kim, D., Dai, Y., Yang, L.T. (eds.) Embedded and Ubiquitous Computing – EUC 2005 Workshops. LNCS, vol. 3823, pp. 652–662. Springer, Heidelberg (2005)
34. Yoneki, E., Bacon, J.: Openness and Interoperability in Mobile Middleware. CRC Press, Boca Raton (2006)
35. Zhao, B.Y., et al.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications 22 (2004)
36. Zhuang, S.Q., et al.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Proc. ACM NOSSDAV, pp. 11–20 (2001)